EXCERPT FROM THE

PROCEEDINGS

OF THE

FIFTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

ReSEARCH: A REQUIREMENTS SEARCH ENGINE

Published: 23 April 2008

by

Dr. Craig Martell, LT Paige H. Adams, Dr. Pranav Anand, ENS Grant Gehrke, Dr. Ralucca Gera, CPT Marco Draeger, Dr. Kevin Squire

5th Annual Acquisition Research Symposium of the Naval Postgraduate School:

Acquisition Research: Creating Synergy for Informed Change

May 14-15, 2008

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **23 APR 2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **ReSEARCH: A Requirements Search Engine**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Computer Science,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
5th Annual Acquisition Research Symposium: Creating Synergy for Informed Change, May 14-15, 2008 in Monterey, CA

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**53**	

The research presented at the symposium was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request Defense Acquisition Research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
555 Dyer Road, Room 332
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
E-mail: jbgreene@nps.edu

Copies of the Acquisition Sponsored Research Reports may be printed from our website www.acquisitionresearch.org

Conference Website:
www.researchsymposium.org

THIS PAGE INTENTIONALLY LEFT BLANK

ReSEARCH: A Requirements Search Engine

Presenter: Craig Martell holds a PhD from the University of Pennsylvania and is an Associate Professor in Computer Science at the Naval Postgraduate School. He specializes in natural-language processing applied to on-line chat, weblogs and semantic search.

Dr. Craig Martell
Associate Professor
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: cmartell@nps.edu

Author: Paige H. Adams, Lieutenant, is a US Navy Information Warfare Officer. He has served as a Russian and Burmese linguist aboard a variety of surface, subsurface, and airborne platforms in the Atlantic, Pacific, and Arabian Gulf, as well as at joint field sites and on flag staff. Most recently, he served as Operations Officer and Joint Operations Department Head at the Misawa Security Operations Center in Japan. He is a 2000 graduate (summa cum laude) of Hawaii Pacific University with a BS/BA in Computer Information Systems.

LT Paige Adams, USN
Graduate Student
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: phadams@nps.edu

Author: Dr. Pranav Anand is an Assistant Professor of Linguistics at UC Santa Cruz. He specializes in theoretical syntax and semantics, with an emphasis on how contextual factors affect the form and meaning of linguistic expressions. He received his AB from Harvard in 2001 and his PhD from MIT in 2006.

Dr. Pranav Anand
Assistant Research Professor
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: panand@nps.edu

Author: Grant Gehrke, Ensign, is a native of Louisville, Kentucky. He graduated in 2007 from the Naval Academy with a degree in Computer Science. His current research focuses on using NLP approaches for authorship attribution in blogs. He will be graduating from the Naval Postgraduate School in June and reporting to Forrest Sherman, a destroyer homeported in Norfolk, Virginia.

ENS Grant Gehrke, USN
Graduate Student
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: gtgehrke@nps.edu

Author: Dr. Ralucca Gera received her BS (2000), MA (2002), and PhD (2005) in Mathematics, all from Western Michigan University. She has been an Assistant Professor in the Applied Mathematics Department at NPS since 2005. Gera's research interests are in combinatorics and graph theory, particularly domination and alliances in graphs.

Dr. Ralucca Gera
Assistant Professor
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: rgera@nps.edu

Author: Marco Draeger is a Captain in the German Army and studied Computer Science at the University of the Federal Armed Forces from 2000 to 2004. From 2004 to 2005, he was assigned to the Modeling and Simulation Department of the Center for Transformation of the Federal Armed Forces. Following this, he served at the German Command Support School until 2007. Since June 2007, Draeger has been a student in the Operations Research Department at the Naval Postgraduate School.

Marco Draeger
Graduate Student
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: mdraeger@nps.edu

Author: Dr. Kevin Squire received his PhD degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign (UIUC), Urbana, in 2004. He is currently an Assistant Professor of Computer Science at the Naval Postgraduate School, Monterey, CA. His research focuses on developing and applying machine-learning algorithms and models to many different domains—including natural language processing, autonomous language acquisition, scheduling, robotics, computer vision, and remote sensing.

Dr. Kevin Squire
Assistant Professor
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
E-mail: kmsquire@nps.edu

Abstract

This research address three closely related problems. (1) Most current search technology is based on a popularity metric (e.g., PageRank or ExpertRank), but not on the semantic content of the document. (2) When building components in a service-oriented architecture (SOA), developers must investigate whether components that meet certain requirements already exist. (3) There is no easy way for writers of requirements documents to formally specify the meaning and domain of their requirements. Our goal in the research presented here is to address these concerns by designing a search-engine that searches over the "meanings" of requirements documents. In this paper, we present the current state of the ReSEARCH project.

Keywords: Semantic Search, Requirements, Open Architecture, Information Systems Technology

1 Motivation

While modern computing has made it possible to access enormous amounts of information with little effort, much of that information comes without any indexing, making manual search of it all but impossible. The science of *information retrieval* (IR) attempts to correct for this by extracting information from a collection of documents based upon a search request, or *query*. While the field of IR has focused a great deal of attention on how the form, or *syntax*, of a query and the documents in the collection can aid the process of extracting information, it has paid far less attention to the meanings, or *semantics*, of those forms.

Semantic analysis can be computationally intensive, and for certain domains, sensitivity to meaning may not provide a system with sufficient improvement to justify the greater computational cost incurred. However, there are at least two conditions in which semantically sensitive search can lead to improvements over keyword-based approaches: a) when the document collection is composed of human-generated free-text, and b) when the document collection is in a specialized domain, with non-standard terminology and assumptions (where the standard for most IR is the general content of the World Wide Web).

The Software, Hardware Asset Reuse Enterprise (SHARE) repository card catalog is in the intersection of both conditions mentioned above. The SHARE card catalog should ideally allow a user to search for an asset based upon free-text overviews generated during asset submission, as well as additional structured metadata (Johnson & Blais, 2008). Because this overview is written in free-text, the syntactic form in which the information expressed by the overview cannot be guaranteed in advance, making search over it quite difficult. In addition, the elements being searched over are descriptions of military assets. So, the document collection for this IR task is in a specialized domain, and the search process should be sensitive to the semantic connections that are particular to this domain.

In order to appreciate the challenges posed by IR over free-text and in specialized domains, we now turn to the complications that each condition brings to the task.

1.1 Challenges of Free-Text Search

Human language in general has several properties that make information retrieval taxing. Formally speaking, any language, human or man-made, can be expressed as a relation between *form* (syntax) and *meaning* (semantics); thus fluency in a domain consists of knowing the relation between the forms of the language and their corresponding meanings. Man-made languages often aim to make this relation as straightforward as possible. For instance, in the mathematical language of arithmetic the syntactic symbol “+” stands for the semantic concept of numerical addition. However, note that the symbol “−” can stand for two different semantic concepts: numerical subtraction or the marking of negative numbers. Thus, "−" has multiple meanings, and we say that it is *polysemous*. In arithmetic, only "−" is polysemous, but in human languages polysemy is pervasive. The word *tank*, for example, has multiple meanings (or, *senses*)—it may refer to weaponry or a water tank. In the information-retrieval context, polysemy renders a query (and sentences in the document set) ambiguous: if the user is searching for tank specifications, are they asking about water tanks or weaponry?

Polysemy complicates the form-meaning relation by having multiple possible meanings for a given word. In addition, human language routinely has multiple words attached to a given meaning. We call these *synonyms*. For example, the verb *consume* has many synonyms, e.g., *devour, ingest, eat*. If a user enters the query "What type of fuel does an F-22 consume?", without an understanding of synonymy, the system will not be able to return to the user, for example, a document containing the answer "The F-22A Raptor uses JP-8". Hence, synonymy complicates the information-retrieval task by creating the (quite likely) possibility that the meaning requested by the user is expressed in a different form than the one the user used in the query. Synonymy may occur at all levels of linguistic form; for example, the sentences, "The F-22A uses JP-8" and "JP-8 is the fuel type for the F-22A Raptor" convey the same semantic information despite their rather different forms. In particular, the first sentence lacks a synonym for *fuel*, meaning that sentence-level synonymy cannot simply be the product of word-level synonymy.

One final complication of searching over human language is that the relationships between semantic entities are not necessarily represented in the syntactic forms of the entities. For instance, the semantic entities *mother* and *daughter* are connected by a parental relation. In order to determine the sentential synonymy of *Mary is Jane's mother* and *Jane is Mary's daughter*, the system must understand the the relationship between *mother* and *daughter*. This is a rather challenging task if we are simply looking at linguistic form, as there is nothing in the words *mother* and *daughter* that indicates they are connected. Such information is accessible only once we have some representation of the meanings of the words (or larger elements), and some way of deriving inferences between them.

1.2 Challenges of Domain-dependent Search

As detailed in (Johnson & Blais, 2008), the SHARE repository asset library currently consists of combat systems software and supporting artifacts, but will become more diverse (e.g., through the incorporation of hardware components). The card catalog will thus contain information about the specification and function of such artifacts. As Johnson and Blais note, there is (and will continue to be) a high level of similarity between the SHARE artifacts, given that they are all specifiable under the Surface Navy OA Warfare Systems Architecture Element Level Decomposition. Hence, their overviews will share many characteristics atypical of documents found on the Web, making Web-based tools sub-optimal.

However, this speciality of SHARE's domain could prove an advantage for semantically sensitive search. For instance, it could allow for a reasonably robust polysemy control. For example, in the SHARE context, a query involving *consume* is more likely to refer to fuel usage than to eating. Similarly, the domain could aid semantic inferencing (of the sort exemplified by the pair *mother-daughter*) based both on terms in the free-text overview and the larger functional context of a particular asset. Hence, based on facts regarding the objects under discussion within SHARE, the system could conclude that there is a relation between *ballistics* and *shell-size*, allowing searches regarding one to consider documents containing the latter. Additionally, building on the product lines that assets play in the Navy enterprise, the system could infer that a given asset possesses certain properties that may be useful the user.

1.3 Domain-independent Learning for Domain-dependent Rules

Given that the polysemy and inferencing subsystems we are building are particular to the specialized domain of SHARE, one natural question is how such subsystems will be developed? One possibility is to build a handwritten set of rules, and have the IR system look to those rules when performing inferencing, such as that implemented in the Wordnet project.[1] While such strategies are undoubtedly useful, they typically: a) are time-consuming, b) lack empirical coverage (human error may cause a rule to go unnoted), and c) require constant supervision for a dynamic document collection. All three pitfalls are of concern with regard to SHARE; the most troubling is probably the requirement for constant maintenance, given that SHARE is an evolving repository and a potential model for similarly constrained repositories over different kinds of assets.

Given such problems, we propose that the domain-dependent components of ReSEARCH be generated not by human input but by machine learning over the document collection of SHARE and, in the initial stages, additional informational resources. The goal of ReSEARCH is to develop a system of tools for determining domain-dependent resources to address the issues surrounding polysemy, synonymy and inference.

The remainder of this paper will detail the problems of contemporary approaches to IR and our investigations of approaches to integrate semantically sensitive tools. In the following section, we will present an overview of common approaches to IR, and why they will fail in dealing with collections such as SHARE. We will then discuss the algorithmic issues involved in generating tools that allow semantically sensitive searching. Finally, we will present the current status of our implementation of the ReSEARCH system.

[1]Wordnet is an ongoing project directed by George Miller at Princeton University's Cognitive Science Laboratory to encode relations between semantic entities. It may be accessed at http://wordnet.princeton.edu.

2 Prior and Current Strategies

The Web is a tremendously useful repository of information. Unfortunately, this information is unstructured, and there is no canonical "Table of Contents" or "Index," making web search one of the most challenging of today's Internet problems. Two attempts were made to address this challenge: (1) hand-classified directories (as originally used by Yahoo, for example), and (2) query-based search engines (for example, AltaVista and, eventually, Google). This second class is what concerns us here. For more details on Web Search Engines see (Schwartz, 1998).

Search engines employ a centralized architecture in which so-called "spiders" collect website information, and an indexer makes an index of these pages to ease the search. In the early 1990s, the first phase of web search was simply keyword search. In keyword-based search, all pages containing requested keywords are returned, ranked according to the strength of match (e.g., the number of times a word appears in a document, if it is in the title, etc.).

AltaVista used this strategy originally. In 1995, it was the first company to fully index the visible pages on the World Wide Web. Over time, it evolved different search modes: basic search, advanced search, and power search (Notess, n.d.). One advanced feature that AltaVista and other search engines added was stemming (Sapp, 2000). Stemming ensures that words with plurals and suffixes (e.g., -ed, -ing, -er) are always treated as being in their stem form (Hersh, 2003, p. 178). Unfortunately, it is unclear how useful stemming is in the search process (Harman, 1991).

The second phase in Web search was the development of techniques that used the connection between pages to create a ranking of the websites for more accurate search. The indexing problem was changed into finding the most appropriate way to "rank" each website. One easy solution was to make the rank proportional only to the number of other pages linking to the page in question. However, this ranking method turned out to be inaccurate for a variety of reasons. In particular, it did not take into account the source of the links, allowing someone to easily boost the rank of a page by increasing the number of incoming links, thus subverting the indexing mechanism (Langville & Meyer, 2006).

To avoid this index subversion, new methods needed to be developed which took advantage of the link structure of both the Web and the meaning of the queried word so the output was most relevant to the query. The challenge, then, was to increase the relevance of the returned pages to the query itself.

2.1 Page Rank and Expert Rank

In 1998, Google revolutionized search. They did this not by changing the fundamentals, as the pages returned are still those that match the keywords in the query, but by changing the order in which the return pages were presented. Google ranked all pages according to the a then-novel ranking algorithm called PageRank (Brin & Page, 1998).

The essence of the Google innovation is in how the PageRank algorithm works. The rank of each page in a search depends not only on the number of pages pointing to it, but also on the rank and the number of outgoing links of these pages. To further determine the rank of all web pages, Google simulates the behavior of virtual surfers randomly surfing the web. A page's rank is then updated based on how frequently the random surfers visit that page. This pre-existing rank of each individual website is assigned independently of any query. As a result of this ranking, the pages are ranked in order of sociological importance: the more links with higher weight there are to a page, the more important it is in the "society" of pages. Additionally, hubs—pages that have a lot of links pointing to them—are given greater authority. In other words, the importance of a link is determined by both the rank of the linking page and the number of outgoing links from that page. One pitfall of this scheme (which Google attempts to corrects) is that communities of websites can trap random surfers, which in turn, increases the rank of those websites.

As an example of how this is implemented, let "Q" be a query, a list of words, that is saved as a vector $\vec{q}$, whose binary components show whether a particular word is present or not in Q. Also, the information on each website $R_1, R_2, \ldots$ is similarly saved as the binary vectors $\vec{r}_1, \vec{r}_2, \ldots$, respectively. By computing the inner product $\langle \vec{q}, \vec{r}_i \rangle$, $i \geq 1$, or the cosine similarity measure using the normalized vectors corresponding to the vectors above, the system can identify the similarity between the query and the pages in the universe. However, this method does not take into account the correlation between websites and their semantics. To overcome this problem, PageRank uses the *PageRank metric* $PR(P)$ that defines recursively the rank/importance of each page P by

$$PR(P) = (1 - d) + d(\sum_{i=1}^{n} PR(T_i)/C(T_n)),$$

where d is a damping factor (0.15, as used by Brin and Page in (Brin & Page, 1998)), T_i $(1 \leq i \leq n)$ are all the pages pointing to P, and each T_i has $C(T_i)$ outgoing links. So, P receives a fraction of the weight $PR(T_i)$, as this weight is equally spread among all the outgoing links from T_i, for $1 \leq i \leq n$ (Zhang & Dong, 2000).

Ask.com, formerly known as "Ask Jeeves," is another search site offering state-of-the-art search, this time based on technology called "ExpertRank" (Ask.com, n.d.). In addition to examining the number of links entering a site, ExpertRank also attempts to identify topic clusters related to a search, as well as experts within these topics, and use all of this information to rank search results.

2.2 The State of Online Search Using Natural Language Processing

Since the "Semantic Web" has become a buzzword in the Internet community and in business at large, several organizations have emerged to provide "Semantic Search." Many promising companies and research projects have built search systems that crawl the web for annotated data over which to search, such as web sites with RDF data. This search strategy, however, does not allow searching documents that do not have rich, hand-built, metadata. In particular, the vast majority of documents online, written in natural human language, are not searched. A small subset of these search engines, however, have begun tackling the problem of searching documents consisting only of written language, extracting semantic meaning.

Powerset Labs (www.Powerset.com), a San Francisco-based startup, has positioned itself as a forerunner in this field by attempting to leverage natural language processing in their search system. Currently honing their search algorithm, Powerset indexes and searches Wikipedia for question-answering tasks. The documents in this database are written in plain text and, for the purposes of search, do not contain extended metadata. Instead, the Powerset indexing algorithm identifies linguistic features such as *named entities* and *parts of speech* to improve search results.

Being a private, for-profit company, the Powerset search algorithm is not public, but some important functionality can be extracted from public demonstrations. The Powerset labs website currently contains two methods of searching Wikipedia. The first is a general search of the document index, which encourages queries to be phrased as questions. Queries such as "When did earthquakes hit San Francisco?" and "politicians from Virginia" are among the suggested queries. Results of these queries return results that demonstrate term matching on a higher level than keyword search. For example, the Powerset system uses "When" as a wildcard to match dates and times that appear in phrases describing earthquakes in San Francisco. "From" is used, in the second example, to search for phrases that indicate some named entity is "from" Virginia. This improves results significantly over a search with just the keywords "politicians" and "Virginia," as are used in standard search engines.

The search "politicians from Virginia" also reveals that "politicians" matches terms such as "governor" and "senator", indicating that an ontology is used to match the term "politician" with its hyponym, "governor." The search "What do zombies eat?" reveals that the Powerset algorithm also searches over synonyms by returning results containing the synonymous verb "devour." This system does not perform rich disambiguation, however, as evidenced by the result "...zombie finishes college," in which "finishes" is considered a synonym of "eat".

Finally, results from the Powerset search "What do zombies eat" include phrases in which the information about what zombies eat is encoded in more complex sentence structures. Correct results such as "granddaughter eaten by zombies," "zombies ... where they are brought back from the dead by supernatural or scientific means, eat the flesh or brains of the living", and "His corpse is thrown over the fence to be devoured by the zombies", all reveal that powerful parsing of the sentences is performed in the indexing process rather than strictly requiring matching phrases such as "zombies eat *". Though their indexing structure is not known, the "PowerMouse" demonstration allows the user to search the fact index more directly, confirming that these relationships exist in the indexing for fast searching, eliminating the need for computationally expensive parsing with every search query.

Powerset is thus building capabilities for semantically sensitive search similar to those of ReSEARCH. However, it is not clear that Powerset's approach is designed to handle the domain-specificity of collections like SHARE, meaning that it is not clear their technology can be leveraged to construct novel inferencing mechanisms in particular domains.

3 Automated Inference-rule Discovery

Recall that natural languages, unlike formal taxonomic structures, contain inherent ambiguity of both form and meaning. It is this ambiguity that presents a challenge for natural language applications such as information retrieval or question answering. Two questions arise: 1) which meaning of a word or phrase in a search term does the requester intend, and 2) how do we return results that are related to the search query, even if the search term does not contain the exact word or words? The first question is related to the problem of *word sense disambiguation* and is, itself, a well-studied area. We shall turn our attention to the second problem: *inference*.

3.1 Semantic Similarity from Distributional Similarity

In (Hearst, 1992), Hearst explored using one kind of inference rules to generate others, given a body of text. Specifically, she considered how use of synonymy relations could be used to learn the relation of *hypernymy*, or subtype classification. For concreteness, consider the pair *vehicle-Humvee*. As *Humvee* is a subtype of *vehicle*, the latter is a hypernym of the former. How could a machine learn the hypernymy relation of *vehicle-Humvee* automatically? Hearst's approach exploited the fact that the co-occurrence of words in patterns of the type *X such as Y*, as well as its synonyms *X, including Y,* and *Y and other X*, implies a hypernymic relationship between *X* and *Y*. As she demonstrated, if a system were seeded with various synonyms for forms that demonstrate hypernymy, the system could induce hypernymic

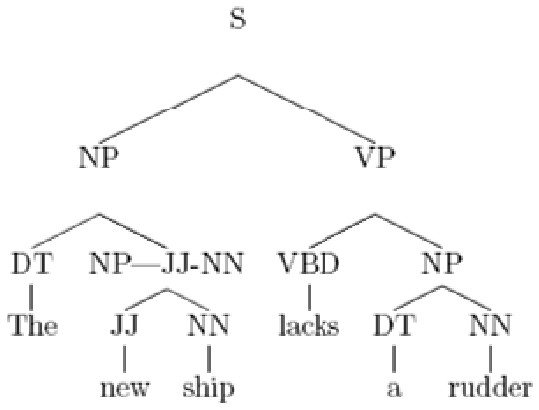


Figure 1. Parse Tree from NLTK Demo

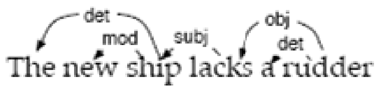


Figure 2. Dependency Tree of Same Sentence as in Figure 1

connections from the text provided.

While Hearst's method is useful for learning various inferences, it relies upon human-generated synonyms for expression of hypernymy (or the relation in question). More desirable would be a system that learns the synonyms themselves from the text, especially given the possibility that such synonyms could be domain-dependent. In their 2001 study, "DIRT—Discovery of Inference Rules from Text"(Lin & Pantel, 2001), Lin and Pantel outline an unsupervised method of discovering inference rules from text, based on the idea that semantic similarity is generally correlated with syntactic similarity. We turn to this next.

3.2 Dependency Trees and Paths

A *dependency* relationship is an asymmetric binary relationship between two words: a **head** and a **modifier**. One can observe the structure of a sentence by examining the tree formed of the dependency relationships contained therein. The tree structure arises from the characteristic that a given word may have more than one modifier, but each word may modify a maximum of one word. Note that a dependency tree differs from a parse tree, which is concerned with the *syntactic* relationship between words. A comparison of the two are shown in Figures 1 and 2.

Dependency graphs are constructed by using Lin's MINIPAR, a broad coverage English language dependency parser (Lin, 2008). Links in the graph represent indirect *semantic* relationships between two words. A dependency path is constructed by joining the words and their link dependency relationships, excluding the two end words. For instance, in our example sentence the dependency path between the words *ship* and *rudder* would be represented by the path N:subj:V←lacks→:V:obj:N. The words *ship* and *rudder* fill the slots in the path at either end. Non-slot dependency relations are called internal relations. In this manner, one can construct the paths of all word pairs in a given corpus of text.

Lin and Pantel (Lin & Pantel, 2001) imposed a set of constraints on the paths to be extracted:

- The "slot fillers" must be nouns, since these are variables that will be instantiated by entities.
- Dependency relations that do not connect the two content words (e.g., in the case of determiners or modifiers), will be excluded from the path.
- There will be a lower limit (threshold) on the frequency count of an internal relation.

To accumulate the frequency counts of paths in a corpus, a **triple database** was used. A triple is comprised of $(p, Slot, word)$ for two words w_1 and w_2. Correspondingly, each such pair of words has two corresponding triples: $(p, SlotX, w_1)$ and $(p, SlotY, w_2)$. $SlotX, SlotY$ and w_1, w_2 are *features* of path p.

3.3 Path Similarity

As alluded to above, Lin and Pantel's approach makes an assumption based on Harris's *Distributional Hypothesis* (Harris, 1954), which assumes that two words will have a similar meaning if they appear in similar contexts. Instead of words, Lin and Pantel assume that the hypothesis also holds for paths between words; i.e., if multiple dependency tree paths link the same set of words, then the meanings of the paths are likely similar. They termed this the *Extended Distributional Hypothesis.*

Computing similarity between two paths first takes into account the **mutual information** between a path slot and its filler. The approach is similar to calculating a $tf \cdot idf$(*term frequency* × *inverse document frequency*) measurement and is performed for a similar reason: to discount high frequency words that may not have the same importance as less frequent words. Pantel and Lin's formula leverages the similarity measurement proposed in (Lin, 1998), but is modified to take paths into account:

$$mi(p, Slot, w) = \log \frac{|p, Slot, w| \times |*, Slot, *|}{|p, Slot, *| \times |*, Slot, w|}.$$

The mutual information thus defined, the similarity between a pair of slots is defined as:

$$sim(slot_1, slot_2) = \frac{\sum_{w \in T(p_1,s) \cap T(p_2,s)} (mi(p_1, s, w) + mi(p_2, s, w))}{\sum_{w \in T(p_1,s)} mi(p_1, s, w) + \sum_{w \in T(p_2,s)} mi(p_2, s, w)}.$$

In this formula, p_1 and p_2 are paths, s is a slot, and $T(p_i, s)$ is the set of all words that fill the s slot of path p_i. Finally, the similarity of two paths p_1 and p_2 is defined by the geometric average of the similarities of their $SlotX$ and $SlotY$ slots:

$$S(p_1, p_2) = \sqrt{sim(SlotX_1, SlotX_2) \times sim(SlotY_1, SlotY_2)}.$$

Comparison of paths in a corpus is accomplished via pairwise comparison of each path using the preceding formulae. Since comparison of all paths is computationally expensive, Lin and Pantel use a filtering algorithm that only compares paths where a candidate path's shared features with an input path p exceed a fixed percentage. This procedure ultimately produces a list of paths in descending order of their similarity to p.

3.4 Results

Lin and Pantel (2001) used MINIPAR to parse approximately 1GB of newspaper text from the *AP Newswire*, *San Jose Mercury-News*, and *The Wall Street Journal*. From this, they extracted seven million paths, 231,000 of them unique, which were then stored in a triple database. For evaluation, they used the first six questions of the TREC-8 Question-Answering Track, extracted the paths from the questions, and generated a Top-40 Most Similar list using their algorithm to determine if the generated paths might contain the answer to the questions posed. This output was also compared to a set of publicly available, manually generated paraphrases of the TREC questions. In the evaluation, a path was deemed to be correct if it was likely that the path could generate the correct response to the question, given that the answer could be found in some corpus. An example used by Lin and Pantel (2001) was the path *"X manufactures Y"* generated from the TREC question, *"What does the Peugeot company manufacture?"* One of the Top-40 most similar paths is *"X's Y factory."* Since *"Peugeot's car factory"* is a likely phrase in some corpus, this generated path is classified as correct.

The DIRT algorithm performance varied widely for different paths. It was noted that paths with verb roots tended to perform better than verbs with noun roots since noun root paths tend to occur less often. Lin and Pantel (2001) also found that, even with high-scoring correct paths, there was little overlap between these automatically generated paths and the manually generated paraphrases, suggesting the difficulty for humans in the paraphrase-generation task. As noted earlier in studies of manual inference-rule generation, completeness errors exist due to the difficulty of paraphrase recall for humans. In this capacity, the DIRT algorithm shows promise in augmenting a manual-generation workflow.

Q#	PATHS	MAN.	DIRT	INT.	ACC.
Q_1	X is author of Y	7	21	2	52.5%
Q_2	X is monetary value of Y	6	0	0	N/A
Q_3	X manufactures Y	13	37	4	92.5%
Q_4	X spend Y	7	16	2	40.0%
	spend X on Y	8	15	3	37.5%
Q_5	X is managing director of Y	5	14	1	35.0%
Q_6	X asks Y	2	23	0	57.5%
	asks X for Y	2	14	0	35.0%
	X asks for Y	3	21	3	52.5%

Table 1. A Summary of Lin and Pantel's DIRT Algorithm Results on TREC-8 Questions.

A summary of DIRT results on the TREC data is in Table 1. The column labeled "Man." indicated the number of manual paraphrases generated for the question. The next column shows the number of paths found by the DIRT algorithm. The intersection of those two is in the fifth column. The final column shows the evaluated accuracy of the automatically generated paths.

3.5 Related work

Snow et al. (Snow, Jurafsky, & Ng, 2005) leverage a similar method of automated inference-rule discovery using dependency paths in a continuation of the hypernym discovery method pioneered by Hearst. This method involved using the dependency paths in a feature count vector and conducting a binary classification of hypernymy for word pairs based on vector-distance measurement. The results obtained represented a 16% F-score improvement over previous models, and a 40% improvement when augmented with **coordinate terms** (i.e., terms that share a common hypernym ancestor).

4 Implementation issues

Lucene Java is an Open Source project, available under the Apache License, which provides an accessible API for the development of search applications. Lucene provides plenty of opportunities to construct a semantic search engine. A good overview and documentation is available from the Apache Lucene website (*Lucene-java Wiki*, 2008).

A search application developed with Lucene consists of the same two major components mentioned in Section 2: an indexer and a searcher. The indexer builds an index of the given documents; the structure and content of this index depends on the implementation of the indexer application. Typical contents would be the title of a document, its path, a URL, or the actual text content. Content can be stored in different ways, depending on if it has to be searchable or not. The search application typically converts a search string given by the user into a query and then searches the index for matching items. Later in this section, two short examples will demonstrate these processes.

4.1 Interesting Features of the Lucene API

One remarkable property of Lucene is its flexibility. By overriding the stemming and analyzing algorithms, its behavior can be changed into something completely new, particularly from a keyword search engine into a semantic search engine similar to Powerset; however, a very useful property of Lucene is its accessibility from different environments, e.g., Python.

PyLucene is a Python extension for accessing Java Lucene. This extension allows developers to implement some functionality of the desired application using NLTK, a widely used Python-based project for natural language processing. Documentation and implementation samples for PyLucene can be found in Vajda (2005).

Another very helpful feature is a package for indexing and query expansion based on WordNet synonyms. Using the WordNet application, this package creates a synonym index of words and converts search strings into queries which can be used by Lucene. For our first tests, we built an index of synonyms from WordNet and used it to expand and convert search strings into Lucene-compatible queries.

4.2 The Wikipedia Corpus

For our experiments we decided to use downloadable Wikipedia content (`http://download.wikimedia.org/enwiki/20080312/enwiki-20080312-pages-articles.xml.bz2`).

The size of this file is about 60 GB. This size requires an event-based parser such as SAX. For the first experiments only about 160 MB (more than 12,000 articles) from a partial download were used.

The structure of the XML file is as follows: every article is stored in a `<page>` node, which has several child nodes. From these child nodes we used the `<title>` and `<text>` fields. The special syntax of a Wikipedia page was ignored at first, meaning that all the content of an article was given the same priority–particularly we did not distinguish between headings, links or normal text.

Parsing and indexing 12,738 articles took about four minutes on a Windows Vista PC with an AMD64 CPU and 1 GB memory under non-benchmark conditions.

4.3 Sample Implementations

Two sample implementations will be introduced: a Wikipedia indexer and a small search application.

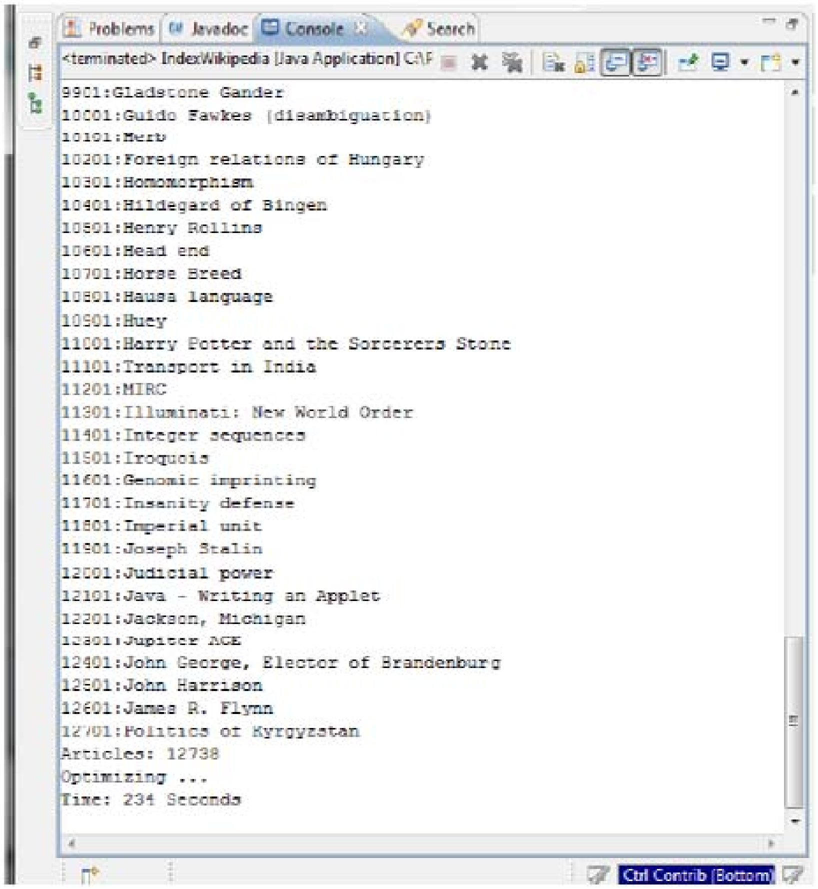
The indexer follows a sample given in (Schmidt, 2005). The original version had to be changed in order to obtain compatibility to the current version of Lucene. Only the main concepts will be considered at this point; for further explanations of the different classes involved, see (*Lucene-java Wiki*, 2008) and Lucene's Javadoc. The main part of an indexing application is the index writer. It writes the index into a file system and also optimizes its structure for faster access. Logically, the written index consists of documents; in our case, every Wikipedia article is treated as a separate document. A document is then split into different fields; for the sample application, these fields were "title" and "text." The indexer determines whether and how a field is stored in the index. The choices are: (1) not to store at all, (2) to store, but not to index, (3) to store and to index it without first analyzing it, and (4) to store it and to index it using an analyzer. An analyzer implements a certain policy for extracting index terms from text. Lucene already implements various analyzers; we used a StandardAnalyzer for our tests. Since an analyzer determines how the content of a document is represented in the index, it provides an opportunity for the developer to implement semantic strategies for building and searching the index. The Wikpedia indexer first parses the xml file, which contains the articles and extracts all `<page>`-nodes, from which the `<title>` and the `<text>` nodes are extracted. Then, for every article, two new fields are generated: "title" and "text." These fields are added to a new document, which is then passed to the IndexWriter-object. After this process, the index content is optimized by the writer, concluding the indexing process.

The searcher was implemented in Python using PyLucene, using a StandardAnalyzer. To be able to search for an article, the user's search string has to be converted into a query. This conversion is done by a Lucene class called QueryParser, which is generated using the name of the field that contains the actual content and the analyzer. The query is then passed to the searcher, which returns an object called "hits." This object holds a list of all matching documents with an assigned score. For our purposes, the searcher application just prints the titles of the matching articles, followed by their score.

The score is assigned by an object which extends the Scorer class in the API. The scorer itself uses a similarity implementation which is based on the cosine-distance between document and query vectors in a Vector Space Model of Information Retrieval.

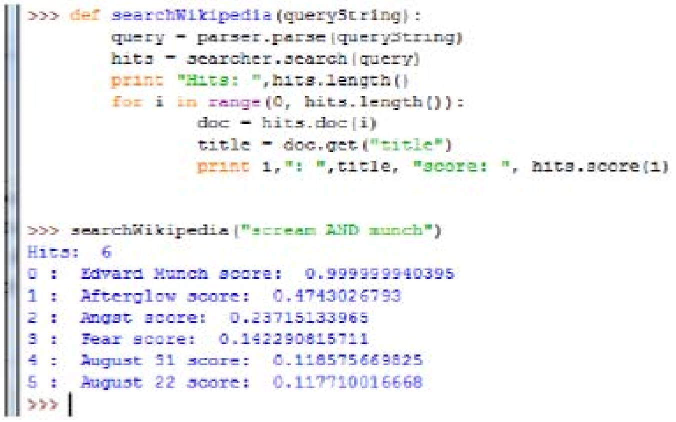
4.4 WordNet Query Expansion Sample

Figure 4.4 shows the output of a standard keyword search using only a single word in the search string versus the output of the same search after expanding the search with the WordNet interface. Only results with a score higher than .5 were printed. This very simple sample



```
<terminated> IndexWikipedia [Java Application] C:\P
9901:Gladstone Gander
10001:Guido Fawkes (disambiguation)
10101:Herb
10201:Foreign relations of Hungary
10301:Homomorphism
10401:Hildegard of Bingen
10501:Henry Rollins
10601:Head end
10701:Horse Breed
10801:Hausa language
10901:Huey
11001:Harry Potter and the Sorcerers Stone
11101:Transport in India
11201:MIRC
11301:Illuminati: New World Order
11401:Integer sequences
11501:Iroquois
11601:Genomic imprinting
11701:Insanity defense
11801:Imperial unit
11901:Joseph Stalin
12001:Judicial power
12101:Java - Writing an Applet
12201:Jackson, Michigan
12301:Jupiter ACE
12401:John George, Elector of Brandenburg
12501:John Harrison
12601:James R. Flynn
12701:Politics of Kyrgyzstan
Articles: 12738
Optimizing ...
Time: 234 Seconds
```

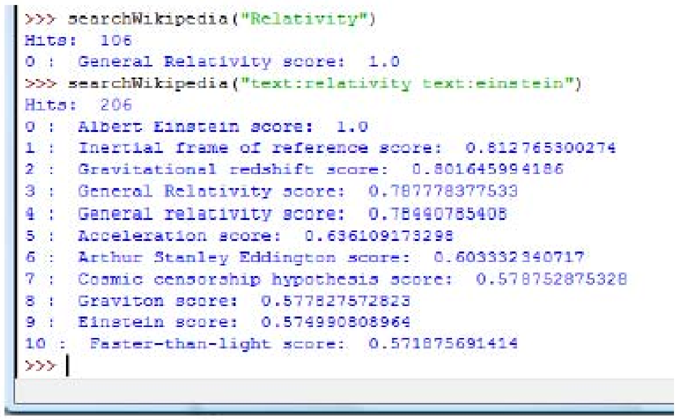
Figure 3. Output of the Wikipedia Indexer



```
>>> def searchWikipedia(queryString):
        query = parser.parse(queryString)
        hits = searcher.search(query)
        print "Hits: ",hits.length()
        for i in range(0, hits.length()):
                doc = hits.doc(i)
                title = doc.get("title")
                print i,": ",title, "score: ", hits.score(i)


>>> searchWikipedia("scream AND munch")
Hits:  6
0 :  Edvard Munch score:  0.999999940395
1 :  Afterglow score:  0.4743026793
2 :  Angst score:  0.23715133965
3 :  Fear score:  0.142290815711
4 :  August 31 score:  0.118575669825
5 :  August 22 score:  0.117710016668
>>> 
```

Figure 4. Searcher Output



```
>>> searchWikipedia("Relativity")
Hits:  106
0 :  General Relativity score:  1.0
>>> searchWikipedia("text:relativity text:einstein")
Hits:  206
0 :  Albert Einstein score:  1.0
1 :  Inertial frame of reference score:  0.812765300274
2 :  Gravitational redshift score:  0.801645994186
3 :  General Relativity score:  0.787778377533
4 :  General relativity score:  0.78440785408
5 :  Acceleration score:  0.636109173298
6 :  Arthur Stanley Eddington score:  0.603332340717
7 :  Cosmic censorship hypothesis score:  0.578752875328
8 :  Graviton score:  0.577827572823
9 :  Einstein score:  0.574990808964
10 :  Faster-than-light score:  0.571875691414
>>> |
```

Figure 5. Comparison of Results Using a Standard Versus an Augmented Search String

shows how using synonyms can improve a search significantly. Note, the Wikipedia article "Relativity" does not appear although it should do so with a score of 1.0. The explanation for this phenomenon is quite simple: the article is not in the corpus–because all experiments were applied on only 12,000 articles, which is less than 1.7% of the actual corpus. To get a perfect hit by a single keyword is, therefore, very unlikely.

4.5 Open Issues

The Lucene API provides several access points through which it can be extended to a semantic search engine. Future work will determine how a document has to be represented in an index to enable a semantic search. This will involve implementation of an analyzer representing the policy for extracting index terms from the corpus. In order to match queries against documents, the analyzer will need to transform search strings into a representation compatible with that of the documents in the index. Additionally, in order to rank documents matching the query according to a scale of relevance, we will need to implement a semantically sensitive scorer.

A final issue of research is how to use WordNet for query expansion beyond addition of synonyms. One relation between words that is worth considering is certainly the hypernym-hyponym relation. WordNet already provides a definition for this relation as a Prolog file. Therefore, a parser for the different WordNet files should be included in the implementation.

5 Conclusion

The ReSEARCH project is still in its beginning stages. However, we have made great strides in identifying the fundamental issues involved in semantic search and how we will need to deal with them in the context of SHARE. Our next step is to start experimentation with proxy data–the Wikipedia data referred to above–and to plan how to move towards *live* SHARE data as it becomes available. Another important aspect of the project that must be handled next is what the *summary* field of the SHARE card catalog must contain.

List of References

Apache Lucene. (2008). *Lucene-java wiki*. Retrieved March 23, 2008, from http://wiki.apache.org/lucene-java

Ask.com. (2008). Ask search technology. Retrieved March 28, 2008, from http://about.ask.com/en/docs/about/ask technology.shtml

Brin, S., & Page, L. (1998, April). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international world wide web conference* (pp. 107-117). Published in *Computer Networks and ISDN Systems*, *30*(1-7).

Harman, D. (1991). How effective is suffixing? *Journal of the American Society for Information Science, 42*(1), 7-15.

Harris, Z.S. (1954). Distributional structure. *Word, 10*, 146-162.

Hearst, M.A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on computational linguistics* (pp. 539-545). Morristown, NJ: Association for Computational Linguistics.

Hersh, W.R. (2003). *Information retrieval: A health and biomedical perspective.* New York: Springer.

Johnson, J., & Blais, C. (2008). Share repository framework: Component specific and ontology. In *Proceedings of the fifth annual acquisition research symposium.* Monterey, CA: Naval Postgraduate School.

Langville, A.N., & Meyer, C.D. (2006, July). *Google's PageRank and beyond: The science of search engine rankings.* Princeton, NJ: Princeton University Press.

Lin, D. (1998). Extracting collocations from text corpora. In *Workshop on computational terminology* (pp. 57-63). Montreal, Canada.

Lin, D. (2008). *MINIPAR*. Retrieved March 28, 2008, from http://www.cs.ualberta.ca/~lindek/minipar.htm

Lin, D., & Pantel, P. (2001). DIRT—Discovery of inference from text. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 323-328). New York: ACM.

Notess, G.R. (n.d.). *Review of altavista.* Retrieved March 23, 2008, from http://www.searchengineshowdown.com/features/av/review.html

Sapp, G. (2000). *Altavista proffers updated search technology to online businesses.* Retrieved March 23, 2008, from http://www.infoworld.com/articles/ic/xml/00/03/21/000321icalta.html

Schmidt, M. (2005). *Lucene wikipedia indexer.* Retrieved March 23, 2008, from http://schmidt.devlib.org/software/lucene-wikipedia.html

Schwartz, C. (1998, September). Web search engines. *Journal of the American Society of Information Science, 49*(11): 973-982.

Snow, R., Jurafsky, D., & Ng, A.Y. (2005). Learning syntactic patterns for automatic hypernym discovery. In L.K. Saul, Y. Weiss & L. Bottou (Eds.), *Advances in neural information processing systems 17* (pp. 1297-1304). Cambridge, MA: MIT Press.

Vajda, A. (2005). *Pulling java lucene into python: Pylucene.* Retrieved March 23, 2008, from http://chandlerproject.org/PyLucene/Paper

Zhang, D., & Dong, Y. (2000). An efficient algorithm to rank web resources. *Computer Networks, 33*(1-6), 449-455.

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2008 Sponsored Research Topics

Acquisition Management

- Software Requirements for OA
- Managing Services Supply Chain
- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Portfolio Optimization via KVA + RO
- MOSA Contracting Implications
- Strategy for Defense Acquisition Research
- Spiral Development
- BCA: Contractor vs. Organic Growth

Contract Management

- USAF IT Commodity Council
- Contractors in 21st Century Combat Zone
- Joint Contingency Contracting
- Navy Contract Writing Guide
- Commodity Sourcing Strategies
- Past Performance in Source Selection
- USMC Contingency Contracting
- Transforming DoD Contract Closeout
- Model for Optimizing Contingency Contracting Planning and Execution

Financial Management

- PPPs and Government Financing
- Energy Saving Contracts/DoD Mobile Assets
- Capital Budgeting for DoD
- Financing DoD Budget via PPPs
- ROI of Information Warfare Systems
- Acquisitions via leasing: MPS case
- Special Termination Liability in MDAPs

Human Resources

- Learning Management Systems
- Tuition Assistance
- Retention
- Indefinite Reenlistment
- Individual Augmentation

Logistics Management

- R-TOC Aegis Microwave Power Tubes
- Privatization-NOSL/NAWCI
- Army LOG MOD
- PBL (4)
- Contractors Supporting Military Operations
- RFID (4)
- Strategic Sourcing
- ASDS Product Support Analysis
- Analysis of LAV Depot Maintenance
- Diffusion/Variability on Vendor Performance Evaluation
- Optimizing CIWS Lifecycle Support (LCS)

Program Management

- Building Collaborative Capacity
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Terminating Your Own Program
- Collaborative IT Tools Leveraging Competence

A complete listing and electronic copies of published research are available on our website: www.acquisitionresearch.org

Acquisition Research Program:
Creating Synergy for Informed Change

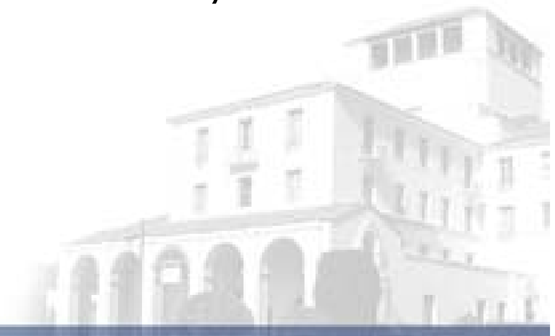
A Semantic-Based Search Engine for Open Architecture Requirements Documents

Dr. Craig Martell

Associate Professor, Naval Postgraduate School

Our Team

- Paige Adams, LT USN
- Pranav Anand, Assistant Prof, Linguistics, UCSC
- Grant Gehrke, ENS USN
- Ralucca Gera, Assistant Prof, Mathematics, NPS
- Marco Draeger, CPT German Army
- Craig Martell, Associate Prof, Computer Science, NPS
- Kevin Squre, Assistant Prof, Computer Science, NPS



The ReSEARCH Project: What we're up to.

- Using open-sourced components, we want to design a semantic search engine for requirements documents that supports the SHARE repository
- We need to match over the meaning of a *requirement*, not a question or a query string.
- Do processing to "enrich" both the query and the documents with semantic information.
- Automatically augment ontologies with new hypernomy ("is a") and mereology ("is a part of") relations.
  - That is, do we have to be told that a Hummer is a vehicle, and one of its parts is a steering wheel, or can we discover this from the text.
- Etc.

Why use semantic search?

- Existing keyword-based search engines do not take into account the semantics of the documents they are searching.

- This is important when trying to find components that do what you need, not what you type.

Why use semantic search?

- The query string and the desired documents may not use the same phrases.

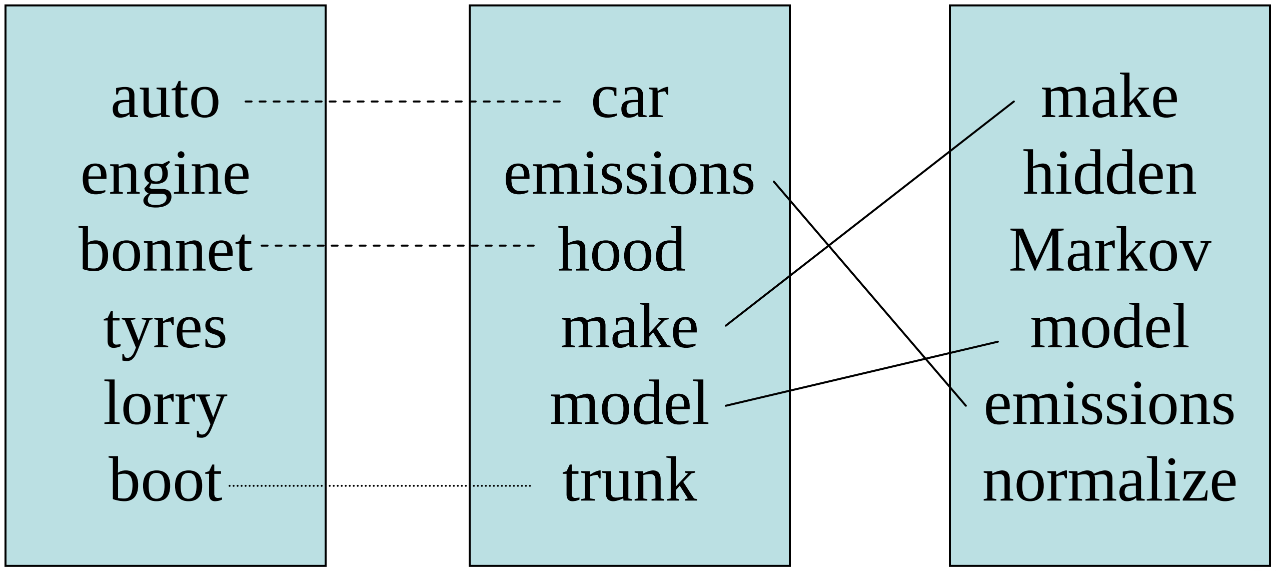
  Q: What fuel does the F-22A consume?

  A: *The F-22A's Raptor uses JP-8.*

- Query and answer convey same meaning, but use different forms
  - Here, “consume” and “uses” are *synonymous.*

Prior and current strategies

Keyword Search Model



Synonymy Problem

Polysemy Problem

Prior and current strategies

- Brin and Page (1998) revolutionized search by using PageRank, which changes the order in which the pages that match the keywords in the query are returned.
- The essence of the Google innovation is in how the PageRank algorithm works.

PageRank algorithm

The rank of a particular page depends on:

- The number of pages pointing to it,
- The rank of each page pointing to it,
- The number of outgoing links on each those pages.

PageRank algorithm

- **PageRank metric** PR(P) defines recursively the rank/importance of each page P by

$$PR(P) \propto \frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \ldots + \frac{PR(T_n)}{C(T_n)}$$

where

- T_1, T_2, ... are all the pages pointing to P
- each T_i has $C(T_i)$ outgoing links.

Random Surfer

To further determine the rank of all web pages Google simulates the behavior of virtual surfers randomly surfing the web.

A page's rank is then updated based on how frequently the random surfers visit that page.

<u>This pre-existing rank of each individual website is assigned independently of any query.</u>

Expert Rank

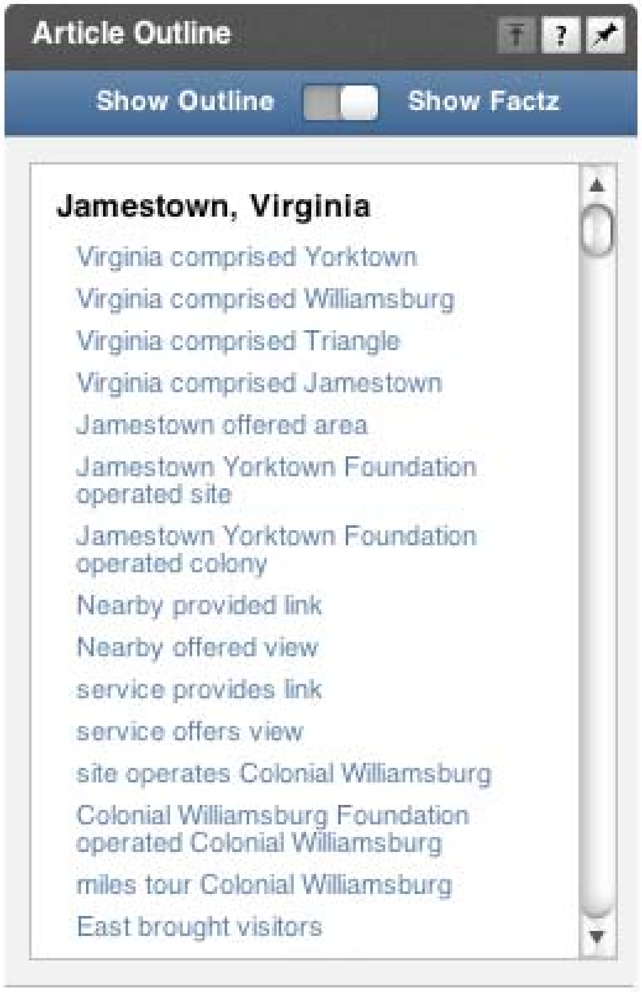
Ask.com (Ask Jeeves) uses the ExpertRank algorithm:

- uses the number of incoming links as well
- attempts to identify topic clusters related to search
- find experts within these topics to “seed” the rank of some websites as “expert” sites.
- PageRank has the problem that “correct” is not the same as “highly-ranked.”

Current Online Semantic Search

- Powerset Labs has emerged as a forerunner in online semantic search using natural language to extract facts from text.
- On 11 May 08, Powerset’s search moved from beta to a public release
- Currently, Powerset searches only Wikipedia documents, but intends to expand search to the Internet in the future.

Powerset Indexing System



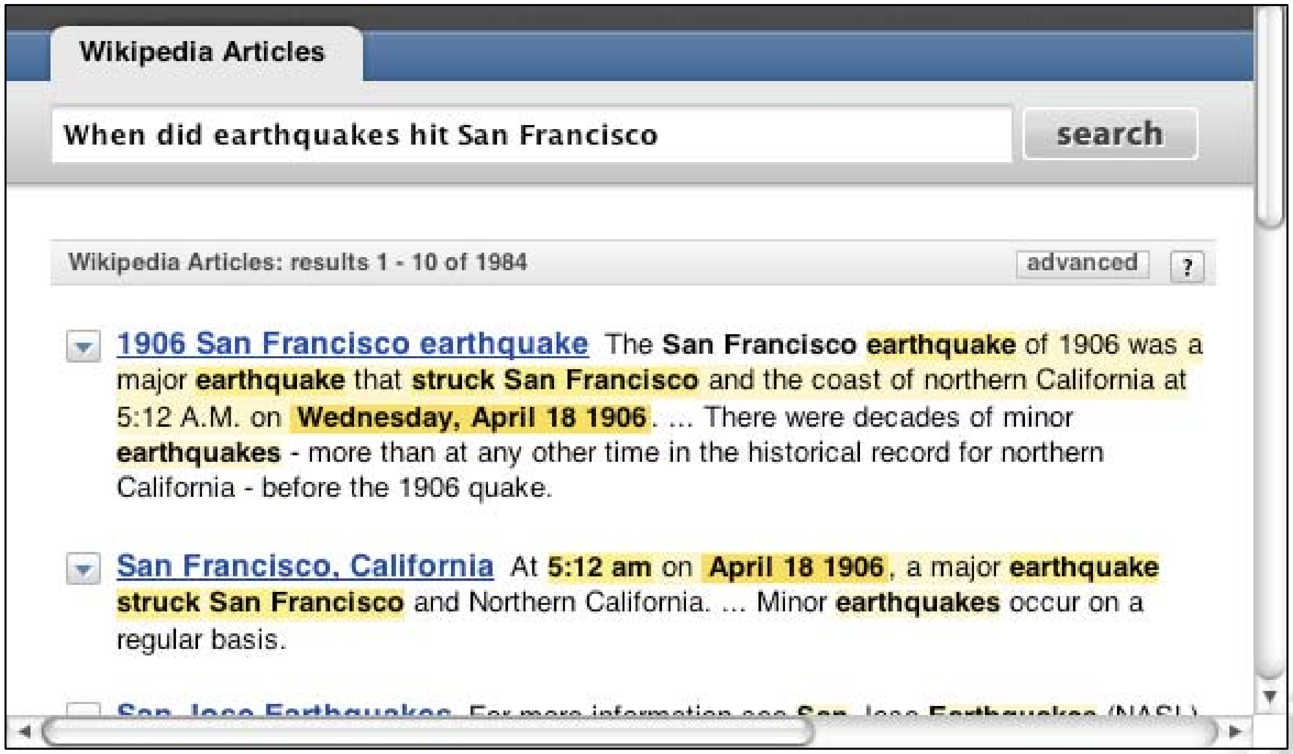
Jamestown, Virginia

Virginia comprised Yorktown
Virginia comprised Williamsburg
Virginia comprised Triangle
Virginia comprised Jamestown
Jamestown offered area
Jamestown Yorktown Foundation operated site
Jamestown Yorktown Foundation operated colony
Nearby provided link
Nearby offered view
service provides link
service offers view
site operates Colonial Williamsburg
Colonial Williamsburg Foundation operated Colonial Williamsburg
miles tour Colonial Williamsburg
East brought visitors

- Powerset's algorithms are not publicly available, but their behavior can be inferred from publicly available demos
  - Powerset parses documents to extract "factz"
  - "Factz" are generally triples of subject-verb-object
  - Search is performed over these "factz" rather than the full text

Question Answering

- Keywords such as “When” tell the system how to narrow results
- “W” words such as “Who” and “When” act as wildcards for matching “factz,” allowing many searches to be matched exactly



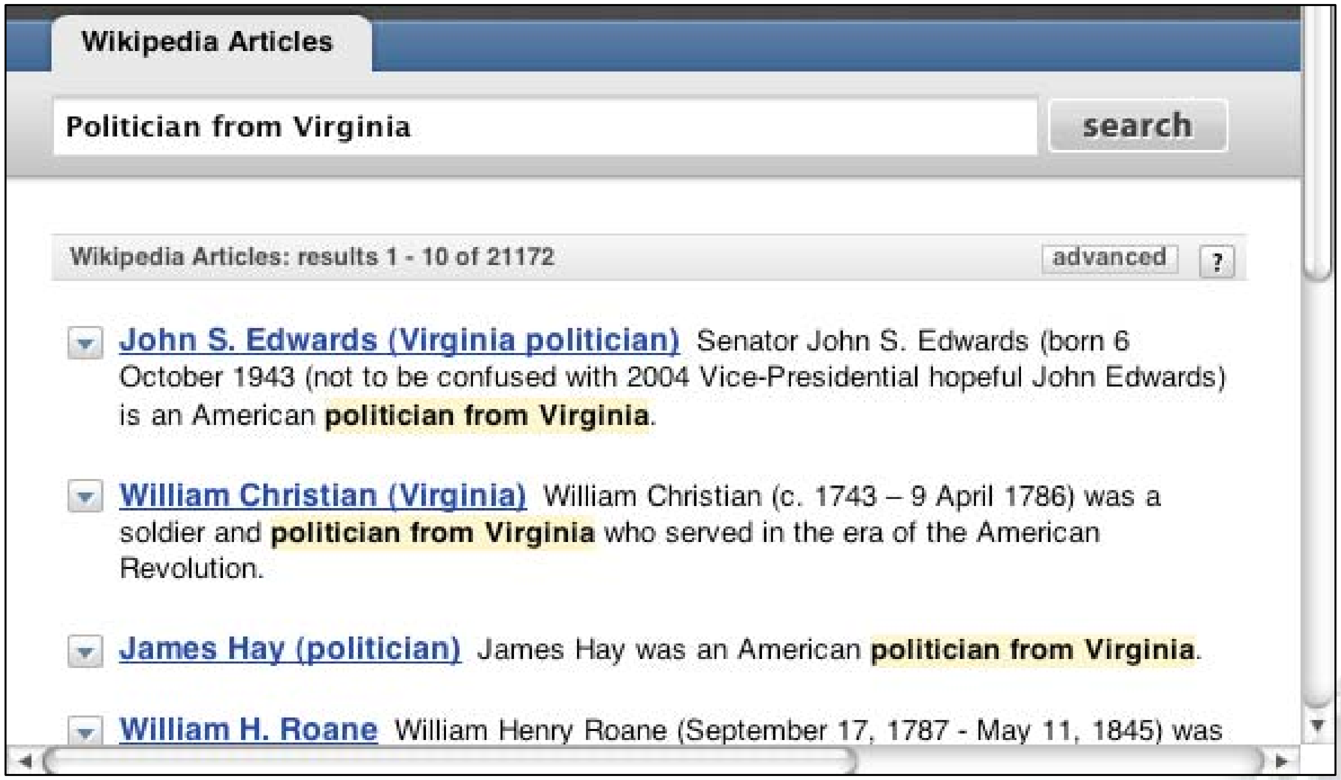
Wikipedia Articles: results 1 - 10 of 1984

1906 San Francisco earthquake The **San Francisco** **earthquake** of 1906 was a major **earthquake** that **struck San Francisco** and the coast of northern California at 5:12 A.M. on **Wednesday, April 18 1906**. ... There were decades of minor **earthquakes** - more than at any other time in the historical record for northern California - before the 1906 quake.

San Francisco, California At **5:12 am** on **April 18 1906**, a major **earthquake** **struck San Francisco** and Northern California. ... Minor **earthquakes** occur on a regular basis.

Question Answering

- Other functional words such as “From” in the search “Politician from Virginia” improve results significantly over searching on just the keywords “Politician” and “Virginia”



Politician from Virginia

Wikipedia Articles: results 1 - 10 of 21172

John S. Edwards (Virginia politician) Senator John S. Edwards (born 6 October 1943 (not to be confused with 2004 Vice-Presidential hopeful John Edwards) is an American **politician from Virginia**.

William Christian (Virginia) William Christian (c. 1743 – 9 April 1786) was a soldier and **politician from Virginia** who served in the era of the American Revolution.

James Hay (politician) James Hay was an American **politician from Virginia**.

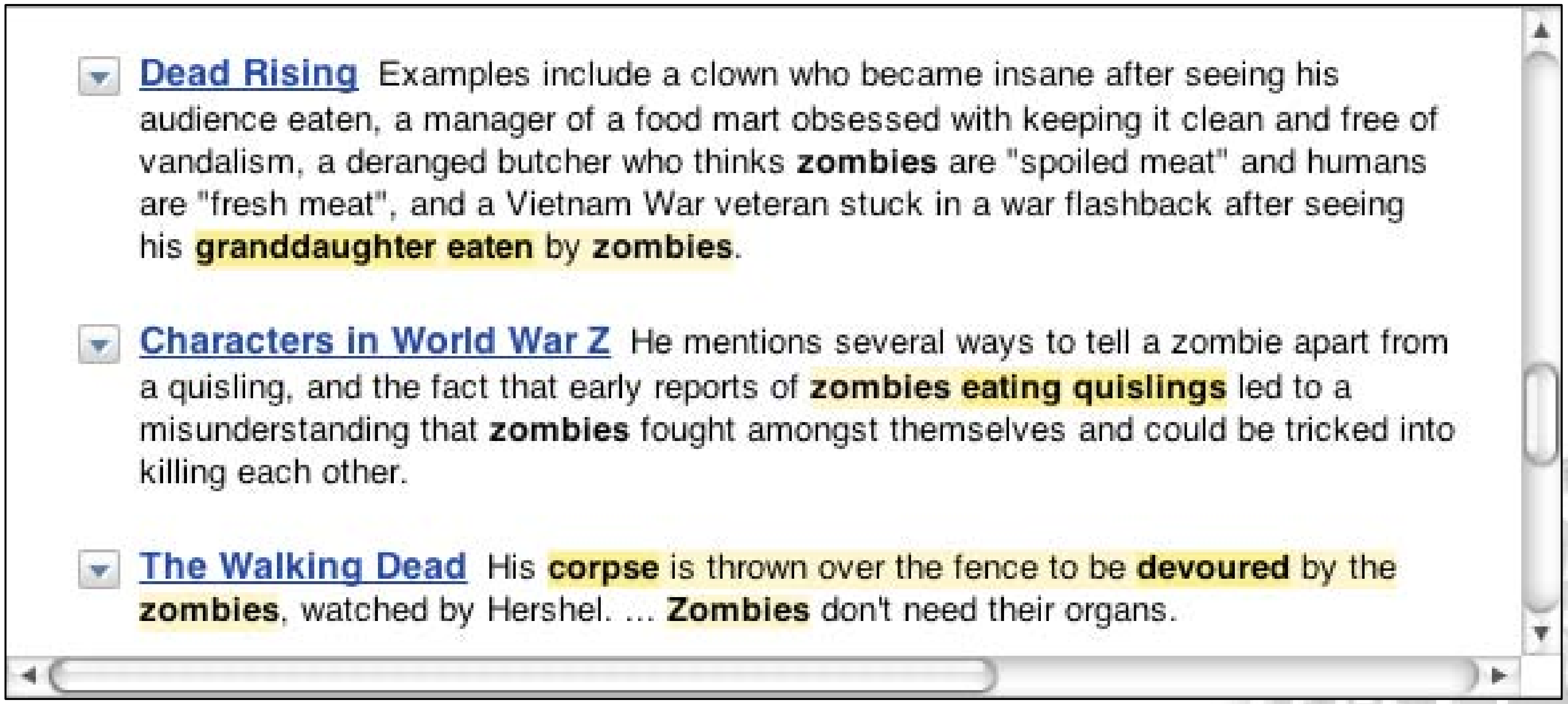
William H. Roane William Henry Roane (September 17, 1787 - May 11, 1845) was

Question Answering

- Question Answering task does not align exactly with requirements document search
- Requirements documents do not hold "Answers" to questions
- Encoding of facts is, however, useful
  – Computationally less demanding
  – Efficient use of storage space for the index
  – Allows *domain specific constructs* of facts to be formulated and recognized in the corpus

Query Expansion with Synonymy

- The search “What do zombies eat?” suggests that Powerset searches for synonyms of query terms, matching “devour”
- Additionally, stemming matches the inverted form “eaten by”


Dead Rising Examples include a clown who became insane after seeing his audience eaten, a manager of a food mart obsessed with keeping it clean and free of vandalism, a deranged butcher who thinks **zombies** are "spoiled meat" and humans are "fresh meat", and a Vietnam War veteran stuck in a war flashback after seeing his **granddaughter eaten** by **zombies**.

Characters in World War Z He mentions several ways to tell a zombie apart from a quisling, and the fact that early reports of **zombies eating quislings** led to a misunderstanding that **zombies** fought amongst themselves and could be tricked into killing each other.

The Walking Dead His **corpse** is thrown over the fence to be **devoured** by the **zombies**, watched by Hershel. ... **Zombies** don't need their organs.

Query Expansion with Synonymy

- Stemming of terms is found in most search engines and is fairly easy to perform
- Matching synonyms allows “close” matches on meaning without requiring an exact keyword match
- Using a structured ontology, expansion is not limited to synonyms but may be extended to hypernyms, hyponyms, and meronyms as well
  - Ontology based query expansion does not appear to be used in current Powerset searches
  - One of our primary approaches:
    - Research Question: Can we automatically augment a given ontology using the text of the documents?

Discovering Synonymous Sentences

- Harris (1954): Synonymous words will occur in the same kinds of environments
- Lin & Pantel (2001): Synonymous sentences will contain the same kinds of words

The F-22A consumes JP-8

The F-22A's engine uses JP-8

- Idea: construct sentence similarity metric

Discovering Synonymous Sentences

- Sentence similarity is the geometric average of the similarity of the **positions** in the sentence:

$$\mathrm{sim}(X_1 \text{ consumes } Y_1, X_2\text{'s engine uses } Y_2) = \sqrt{sim(X_1, X_2) \times sim(Y_1, Y_2)}$$

$$\mathrm{sim}(X_1 \quad p_1 \qquad Y_1, X_2 \quad p_2 \qquad\qquad Y_2) = \sqrt{sim(X_1, X_2) \times sim(Y_1, Y_2)}$$

Discovering Synonymous Sentences

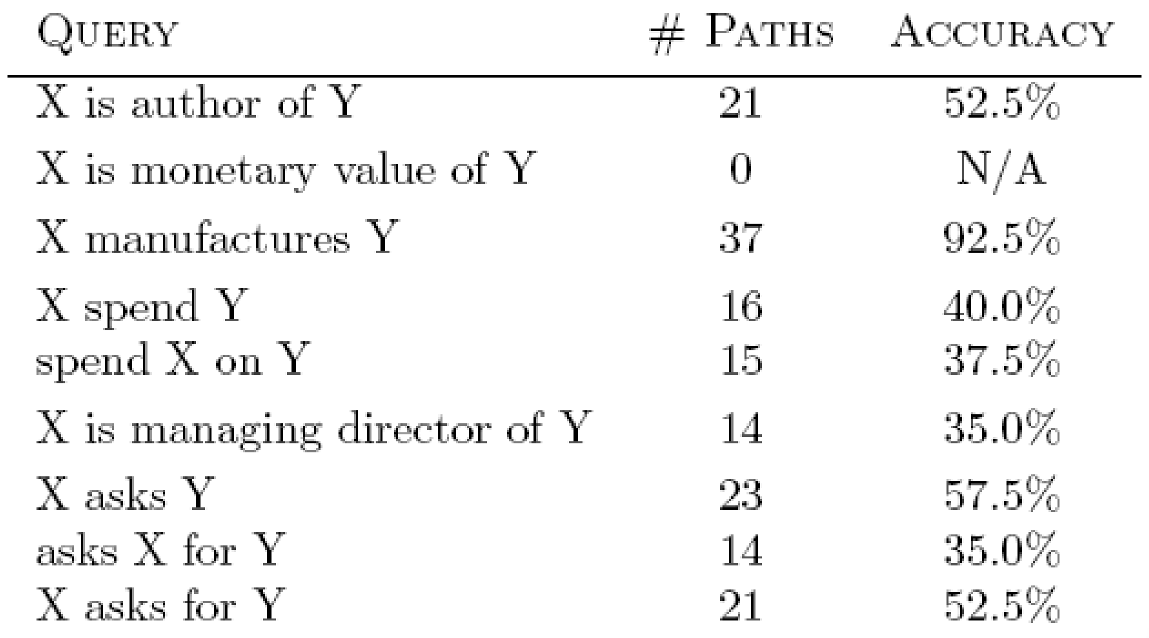
- Position similarity is a normalized sum of the pointwise mutual information of all words that appear in both positions of the respective paths:

$$sim(X_1, X_2) = \frac{\sum_{w \in T(p_1,s) \cap T(p_2,s)} (mi(p_1, s, w) + mi(p_2, s, w))}{\sum_{w \in T(p_1,s)} mi(p_1, s, w) + \sum_{w \in T(p_2,s)} mi(p_2, s, w}$$

$$mi(X_1 \ldots p \ldots Y_1, X_1, w) = \log \frac{\frac{f(p,X_1=w)}{f(*,X_1=w)}}{\frac{f(p,X_1=*)}{f(*,X_1=w)}}$$

Discovering Synonymous Sentences

- Lin & Pantel evaluated system against TREC-8 Question Answering Task question set.



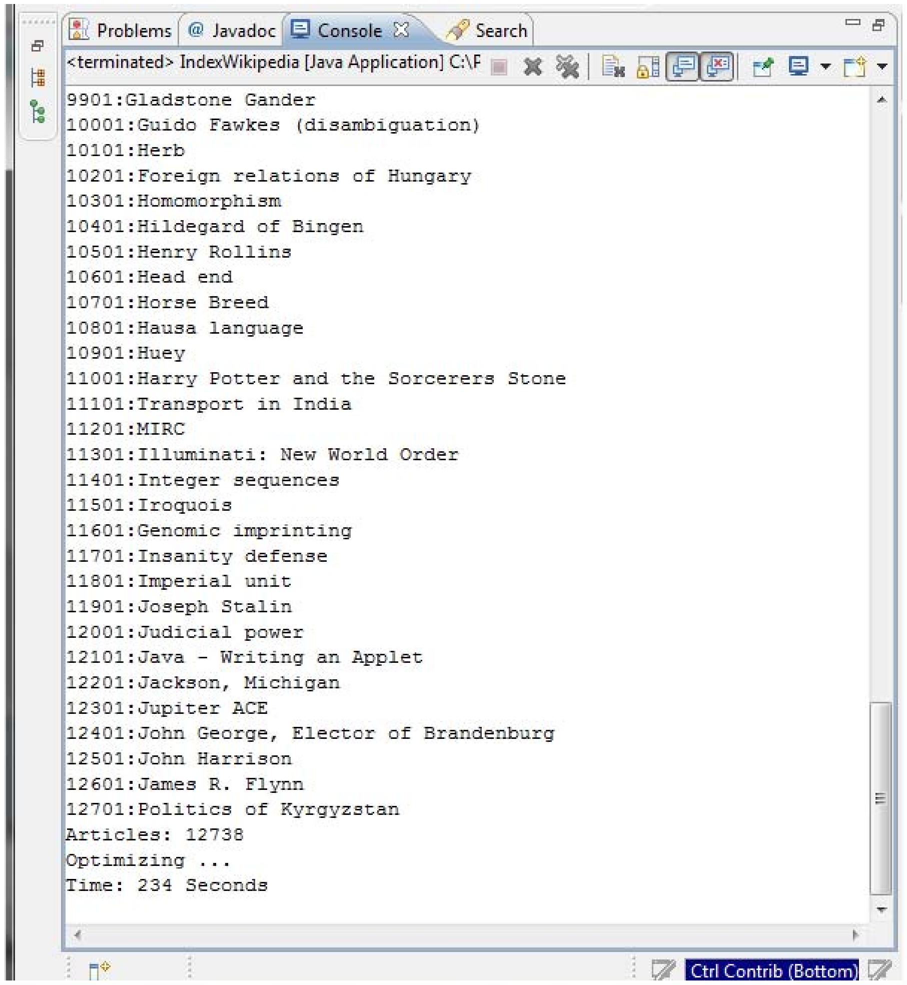
QUERY	# PATHS	ACCURACY
X is author of Y	21	52.5%
X is monetary value of Y	0	N/A
X manufactures Y	37	92.5%
X spend Y	16	40.0%
spend X on Y	15	37.5%
X is managing director of Y	14	35.0%
X asks Y	23	57.5%
asks X for Y	14	35.0%
X asks for Y	21	52.5%

The ReSEARCH Project: Work for us.

- Using open-sourced components, design a semantic search engine for requirements documents that supports the SHARE repository
- Match over the meaning of a *requirement*, not a question.
- Do processing to "enrich" both the query and the documents with semantic information.
- Automatically augment ontologies with new hypernomy ("is a") and mereology ("is a part of") relations.
  - That is, do we have to be told that a Hummer is a vehicle, and one of its parts is a steering wheel, or can we discover this from the text.
- Lots more!

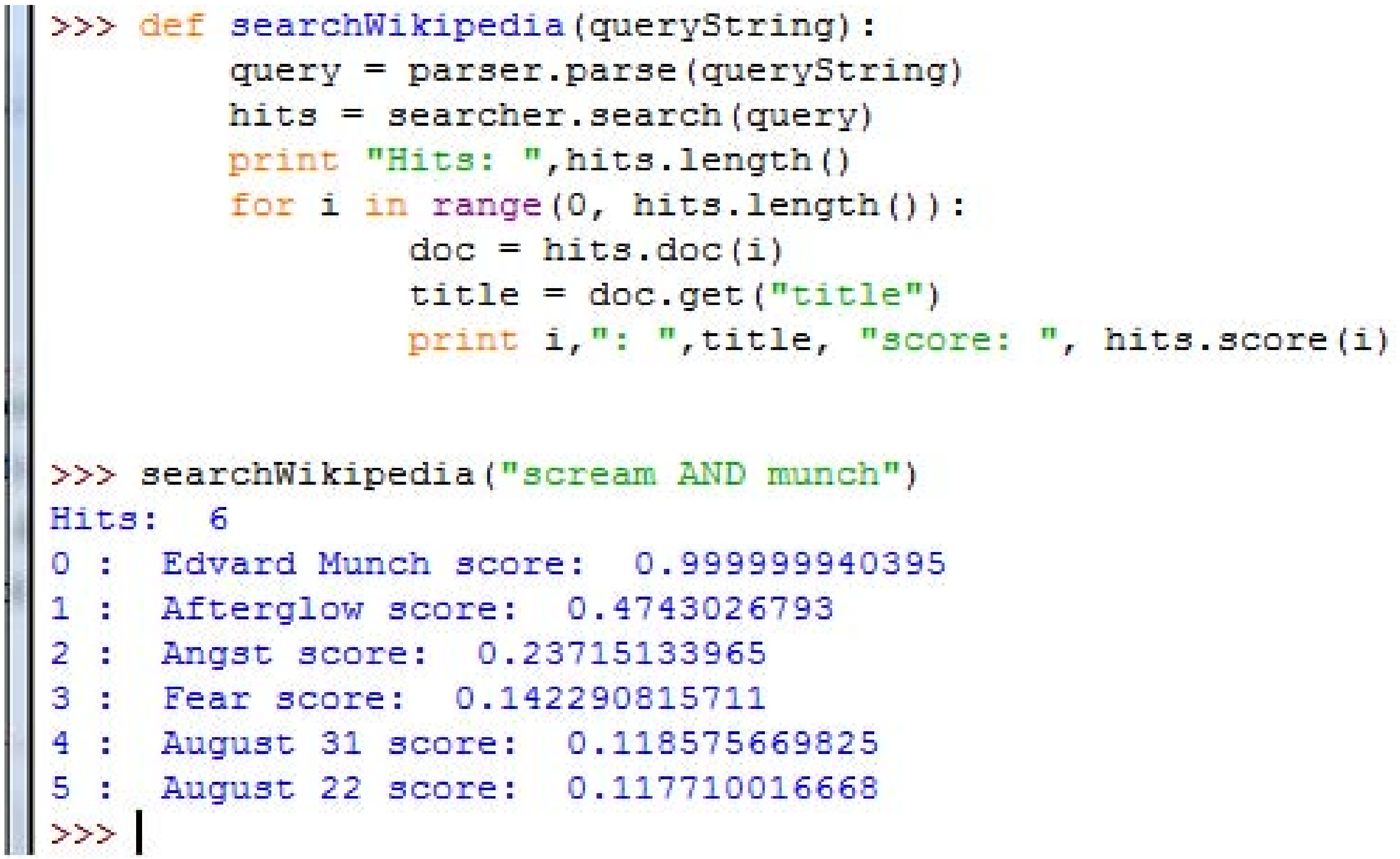
Backup Slides

Building the Index



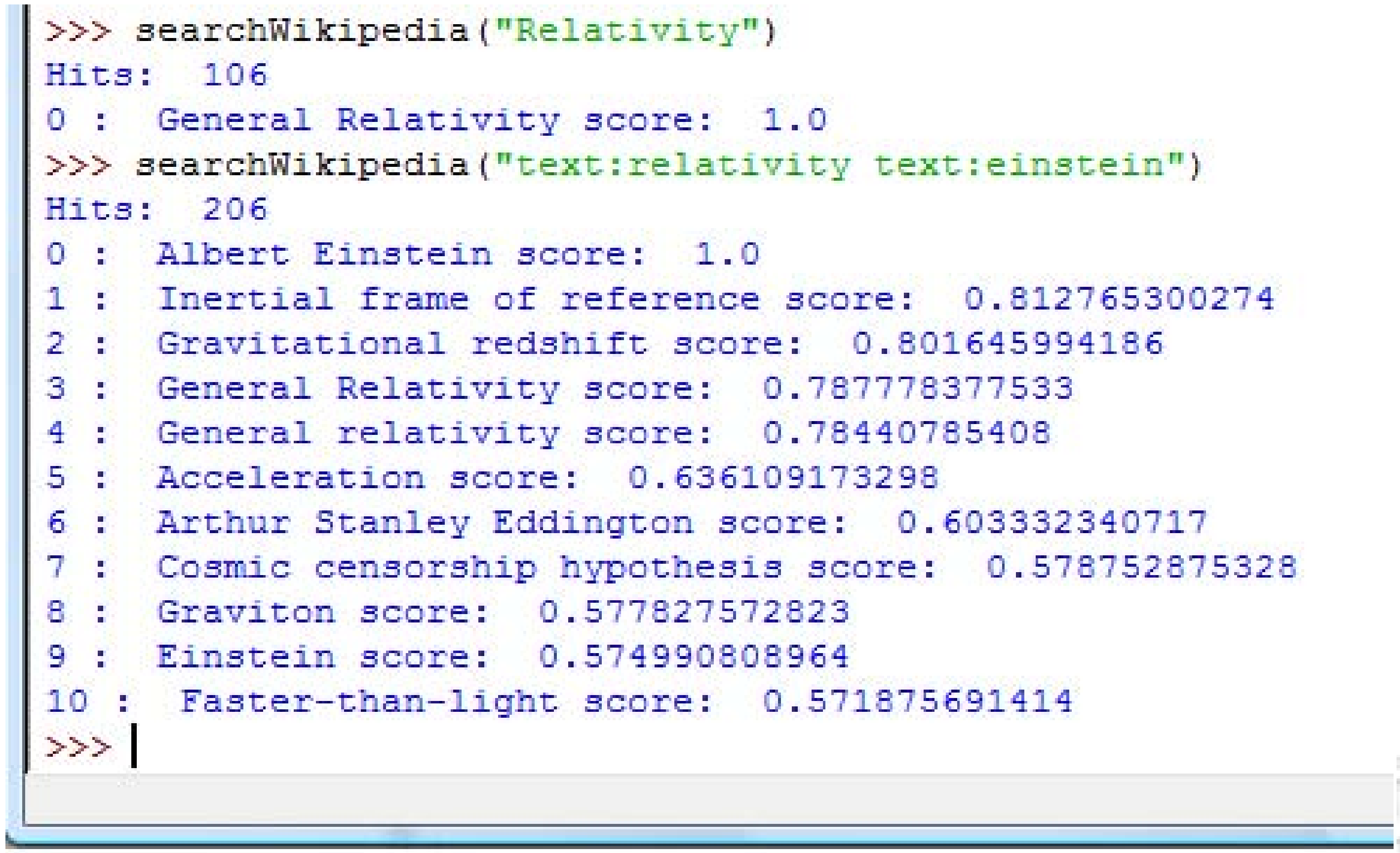
```
<terminated> IndexWikipedia [Java Application] C:\F
9901:Gladstone Gander
10001:Guido Fawkes (disambiguation)
10101:Herb
10201:Foreign relations of Hungary
10301:Homomorphism
10401:Hildegard of Bingen
10501:Henry Rollins
10601:Head end
10701:Horse Breed
10801:Hausa language
10901:Huey
11001:Harry Potter and the Sorcerers Stone
11101:Transport in India
11201:MIRC
11301:Illuminati: New World Order
11401:Integer sequences
11501:Iroquois
11601:Genomic imprinting
11701:Insanity defense
11801:Imperial unit
11901:Joseph Stalin
12001:Judicial power
12101:Java - Writing an Applet
12201:Jackson, Michigan
12301:Jupiter ACE
12401:John George, Elector of Brandenburg
12501:John Harrison
12601:James R. Flynn
12701:Politics of Kyrgyzstan
Articles: 12738
Optimizing ...
Time: 234 Seconds
```

A Sample Search with Lucene


```
>>> def searchWikipedia(queryString):
        query = parser.parse(queryString)
        hits = searcher.search(query)
        print "Hits: ",hits.length()
        for i in range(0, hits.length()):
                doc = hits.doc(i)
                title = doc.get("title")
                print i,": ",title, "score: ", hits.score(i)


>>> searchWikipedia("scream AND munch")
Hits:  6
0 :  Edvard Munch score:  0.999999940395
1 :  Afterglow score:  0.4743026793
2 :  Angst score:  0.23715133965
3 :  Fear score:  0.142290815711
4 :  August 31 score:  0.118575669825
5 :  August 22 score:  0.117710016668
>>> 
```

Using an Augmented Search String



```
>>> searchWikipedia("Relativity")
Hits:  106
0 :  General Relativity score:  1.0
>>> searchWikipedia("text:relativity text:einstein")
Hits:  206
0 :  Albert Einstein score:  1.0
1 :  Inertial frame of reference score:  0.812765300274
2 :  Gravitational redshift score:  0.801645994186
3 :  General Relativity score:  0.787778377533
4 :  General relativity score:  0.78440785408
5 :  Acceleration score:  0.636109173298
6 :  Arthur Stanley Eddington score:  0.603332340717
7 :  Cosmic censorship hypothesis score:  0.578752875328
8 :  Graviton score:  0.577827572823
9 :  Einstein score:  0.574990808964
10 :  Faster-than-light score:  0.571875691414
>>> 
```

Our Work

GOAL: Design an alternative method to explicitly store/represent semantic metadata in order to enable semantic search.